Translittération automatisée des hiéroglyphes égyptiens

Serge Rosmorduc
2008

RÉSUMÉ

Cet article décrit un système qui est capable de donner une translittération raisonnable des textes hiéroglyphiques moyen égyptiens en utilisant un ensemble de « règles de réécriture ». Il donne une brève explication du fonctionnement interne du système, puis il décrit le détail desdites règles.

INTRODUCTION

Il y a quelques années, alors que je travaillais sur l'analyse syntaxique automatisée du *Moyen égyptien* pour mon doctorat, j'ai envisagé la possibilité d'une translittération automatisée[1]. Cependant, comme cette tâche n'était pas centrale dans notre travail à l'époque, je n'ai pas exploré le problème en profondeur. En 1997, j'avais une étudiante, F. Kerboul[2], qui se pencha sur le problème en utilisant le langage informatique *Prolog.* Elle travailla sur l'analyse des mots isolés et, comme les résultats étaient encourageants, je décidai d'approfondir le sujet.

Le système décrit dans cet article prend en entrée le contenu d'un texte hiéroglyphique sous la forme d'une liste de codes du *Manuel de Codage*, et produit en sortie une translittération du texte.

Cet article est écrit avec deux audiences à l'esprit, des égyptologues et des spécialistes du traitement automatique du langage naturel. De ce fait, il contient un certain nombre d'explications assez basiques, tant sur le problème lui-même que sur la méthode utilisée pour le résoudre.

---

[1] S. Rosmorduc. (1996). Analyse morpho-syntaxique de textes non ponctués, Application aux textes hiéroglyphiques . Thèse de doctorat, p. 75; id, 'Traitement automatique du langage naturel en moyen égyptien', PIREI X, Bordeaux, 1994, p. 100–101.

[2] F. Kerboul (1997). Translittération automatique des hiéroglyphes. Rapport de stage de l'ENSTA.

ANALYSE

Translittération des hiéroglyphes égyptiens

Lorsqu'un égyptologue travaille sur un texte hiéroglyphique, il le transcrit en caractères latins qui représentent son squelette consonantique. Dans notre cas, l'entrée consistera en des codes conçus pour décrire les hiéroglyphes, codes définis dans le fameux *Manuel de Codage*[3] (MdC).

Hiéroglyphes			
Code MdC	I10:D46-M17-N35	T18-G43-A1	M17-N29:D21:Y1
Translittération	*ḏd.in*	*šmsw*	*iqr*
Traduction	said	servant	excellent

Le *Manuel de Codage* décrit les textes hiéroglyphiques avec des codes de signes tels que I10 pour  , et des codes de position sous la forme de ':' pour signifier *empiler* (au-dessus). Les valeurs des signes peuvent appartenir à différentes catégories :

- des *phonogrammes*, ou *signes phonétiques*, qui représentent une ou plusieurs consonnes, car les voyelles ne sont pas écrites. Par exemple, le serpent I10 est *ḏ* (comme dans 'djè') tandis que T18 () est *šms* (sh+m+s).
- des *idéogrammes*, qui renvoient directement à leur sens : par exemple,  (E1) peut être utilisé pour écrire le mot *iḫ*, *boeuf*.
- les *déterminatifs*, qui sont une sorte de classificateur sémantique, utilisés au niveau des terminaisons de mots. Dans notre exemple,  (A1) et (Y1, rouleau de papyrus) sont des déterminatifs, respectivement pour les êtres humains et pour les concepts abstraits.

Maintenant, la translittération automatisée n'est pas aussi simple qu'il n'y paraît.

[3] J. Buurman, N. Grimal, M. Hainsworth, J. Hallof et D. v. der Plas Inventaire des signes hiéroglyphiques en vue de leur saisie informatique . Mémoires de l'Académie des Inscriptions et Belles Lettres. Paris : Institut de France (1988)

Premièrement, les signes peuvent avoir plus d'une valeur. Par exemple, le rouleau de papyrus est aussi un idéogramme dans le mot *mḏ3.t*, *document*. Deuxièmement, la combinaison des signes n'est pas simple. Un signe multiconsonantique est souvent accompagné de signes *alphabétiques*, qui correspondent à une partie de son orthographe. Par exemple, Y5,  a la valeur *mn*, mais est souvent accompagné de N35,  , qui est *n*, le groupe Y5:N35 est donc *mn*,  , et non *mnn*.

Approche de base

Pour donner une première idée du fonctionnement de la translittération automatique, commençons par un exemple simple, en considérant le mot  , *3b*, *désirer*. Ce mot est encodé dans le *Manuel de Codage* comme 'U23-D58-A2'. Les signes peuvent avoir la valeur suivante :

1. U23,  , peut correspondre aux consonnes *3b* ou *mr*.
2. D58,  , est la consonne b.
3. A2,  , est un déterminatif pour les actions liées à la bouche.

Connaître les valeurs des signes ne suffit pas. Nous devons également les combiner. En référence aux signes phonétiques ici, deux règles différentes peuvent s'appliquer :

a) on peut considérer qu'un signe bilitère XY suivi d'un signe unilitère Z peut être combiné en un groupe de trois consonnes, XYZ,

b) ou on peut considérer qu'un signe bilitère XY, suivi d'un Y unilitère, c'est-à-dire suivi de sa dernière consonne, peut être combiné en un groupe de deux consonnes, XY.

Dans tous les cas, le signe A2 marque la fin du mot et n'a aucune valeur phonétique. En utilisant toutes les solutions possibles, nous aboutissons à trois interprétations :

1. en utilisant la valeur *mr* pour U23, on ne peut utiliser que la règle a), qui donne la translittération *mrb*.
2. en utilisant la valeur *3b* pour U23, nous pouvons utiliser soit la règle a), produisant *3bb*,
3. soit b), donnant *3b* (la solution correcte).

Ensuite, nous avons besoin d'un moyen de représenter ces règles et d'un moyen de choisir la *meilleure* solution.

Représentation des règles (exemple simple)

Les règles sont représentées comme des *règles de réécriture*, indiquant que, pour une donnée donnée, nous obtiendrons une conclusion donnée. Par exemple, les deux valeurs différentes de U23 seront représentées par :

r1. U23 => P(3,b) / 100
r2. U23 => P(m,r) / 100

où P(3,b) signifie *phonogramme de valeur 3,b*, et 100 est le *coût* de la règle, qui est une mesure de son exactitude. Les valeurs réelles attribuées aux coûts sont plutôt *ad hoc*. Les autres règles pour les valeurs des signes seraient quelque chose comme :

r3. D58 => P(b) / 100
r4. A2 => DET(mouth_action) / 100

où DET(mouth_action) signifie *déterminatif relatif à mouth_action* et 100 le coût de la règle. Maintenant, voici un premier ensemble de règles qui, appliquées à la valeur 'U23 D58 A2', peuvent produire les résultats :

P(3,b) P(b) DET(mouth_action), pour un coût de 300
ou
P(m,r) P(b) DET(mouth_action), également pour un coût de 300.

Un deuxième ensemble de règles servant à combiner les signes est ensuite appliqué. Une première règle précisera qu'un signe unilitère peut être lu seul :

r5. P($X) => L($X) / 100

ici, $X est une variable qui peut être remplacée par n'importe quelle consonne. Nous utilisons deux opérateurs différents, P() et L() pour différencier les valeurs des signes (soit P) et la translittération des mots (soit L).

La même règle est vraie pour les signes bilitères :

r6. P($X,$Y) => L($X), L($Y) / 100

Mais, bien évidemment, il y a la règle qui permet de combiner un signe bilitère et un signe unilitère :

r7. P($X,$Y), P($Y) => L($X), L($Y) / 100

Enfin, on peut envisager une règle stipulant qu'un déterminatif peut marquer la fin d'un mot :

r8. DET($X) => wordend / 100.

Appliquer les règles

Le premier ensemble de règles, concernant les valeurs de signe, est appliqué au texte en entrée. Cela crée un ensemble d'interprétations possibles; puis nous appliquons le deuxième ensemble de règles à ces interprétations.

Le principe est que le *coût* d'une hypothèse est le total des coûts des règles utilisées, et la *meilleure* hypothèse est celle qui a le moindre coût.

Par exemple, pour obtenir la translittération *mrb*, il faut utiliser les règles r2, r3 et r4, puis appliquer ensuite les règles r5, r6 et r8, le résultat étant :

'L(m), L(r), L(b), wordend', avec un coût total de 600.

Pour obtenir la translittération *3b*, les règles utilisées seront r1, r3 et r4, puis r7 et r8, avec un coût de 500. Ainsi, *3b* est meilleur que *mrb*.

Un problème avec ce système est que nous devons en quelque sorte considérer toutes les combinaisons possibles de règles, ce qui, à première vue, est une tâche ardue.

Par exemple, si nous considérons le mot  , (V24-D58-F46-D54), *wḏb*, 'tourner, plier',

 , V24, peut être P(w,D)

 , D58, peut être P(b)

 peut être soit P(p,X,r), ID(w,D,b) ou DET(folding_action)

 peut être ID(i,w), ID(n,m,t,t) ou DET(move)

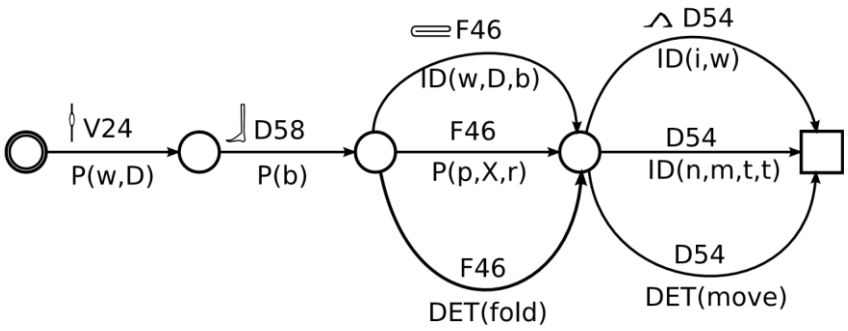
où ID signifie *idéogramme*.

Si nous considérons simplement ces possibilités, nous avons 1×1×3×3, donc neuf interprétations pour les valeurs du signe.

En résumé, le nombre d'hypothèses tend à croître de façon exponentielle avec la longueur du texte[4].

Maintenant, la réécriture des règles est un domaine bien connu en l'informatique, et nous disposons d'une méthode de représentation élégante à la fois des règles et de leur interprétation que l'on appelle *transducteur d'état fini*, lequel est lui-même une extension de la notion d'*automate d'état fini*[5]. La notion est assez simple à appréhender graphiquement.

Voici un transducteur représentant notre séquence de signes :



Sur chaque lien du transducteur, il y a deux informations : l'entrée (au dessus du lien), qui est ce qui peut être lu par le transducteur, et la sortie (en dessous du lien), qui est ce qui peut être produit par le transducteur. L'entrée correspond à la partie gauche de nos règles, et la sortie à la partie droite. Tout chemin entre le nœud le plus à gauche du transducteur et son nœud le plus à droite (le carré) représente un choix d'applications de règles. Ainsi, le transducteur permet la représentation d'un grand nombre d'hypothèses de manière très compacte.

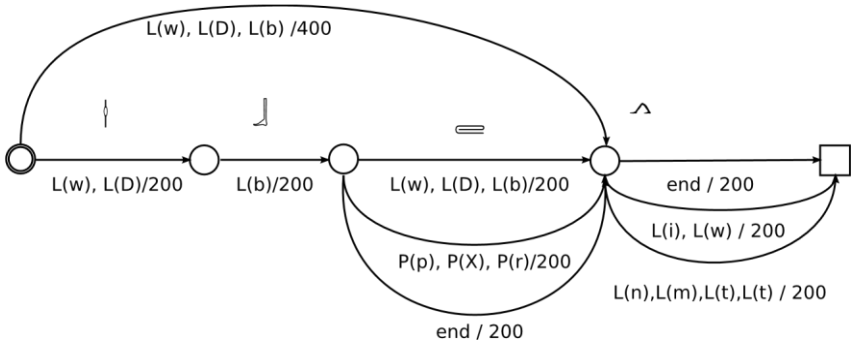
Une autre caractéristique importante des transducteurs est qu'ils peuvent représenter à la fois l'entrée, les règles et leurs résultats.

Considérons quelques règles supplémentaires pour nous permettre d'aller plus loin[6] :

```
ID($A,$B)=> L($A), L($B) / 100
ID($A,$B,$C,$D)=> L($A), L($B),L($C), L($D) / 100
ID($A,$B,$C)=> L($A), L($B),L($C) / 100
P($A,$B), P($B), ID($A,$B,$C) => L($A), L($B), L($C) / 100
P($A,$B,$C)=> L($A), L($B), L($C) / 100
```

[4] Pour donner un autre exemple, si nous avions quatre signes ambigus, chacun avec trois interprétations, le nombre total de combinaisons serait de 3 × 3 × 3 × 3, 81 possibilités !
[5] Pour une couverture théorique et pratique du sujet en relation avec l'informatique et le traitement automatique du langage naturel, voir E. Roche et Y. Schabes, éd. *Traitement du langage à états finis*, MIT Press, 1997.
[6] Ces règles sont en réalité un peu trop simplistes pour permettre de faire l'ensemble de l'analyse, mais l'on souhaite ici souligner les mécanismes du système.

Une fois appliquées, ces règles nous donneront le transducteur suivant :



Chaque trajet dans le transducteur correspond à une analyse possible (bien entendu, seules quelques-unes sont raisonnables). Si nous les comptons, nous constatons qu'il y a maintenant douze possibilités différentes, mais le transducteur les représente toutes avec seulement neuf liens. Une caractéristique intéressante est que les choix qui sont indépendants les uns des autres sont clairement séparés par l'automate.

Problèmes à résoudre

Cela étant dit, il y a un certain nombre de points problématiques à traiter si l'on veut effectuer une translittération automatisée raisonnable d'un texte hiéroglyphique. Nous voulons pouvoir séparer les différents mots. Pour ce faire, nous devons en quelque sorte modéliser la forme d'un mot égyptien.

Sur le plan linguistique d'abord : un mot bilitère ou trilitère est usuel, mais les mots plus longs ont tendance à avoir des formes spécifiques, soit parce qu'ils contiennent des terminaisons grammaticales, soit parce qu'ils dérivent de racines plus simples par adjonction de préfixes ou de redoublements.

Deuxièmement, au niveau graphémique : un mot a tendance à contenir une partie phonétique, suivie d'une terminaison par un déterminatif et diverses marques, comme des indicateurs pluriels. La structure d'une terminaison de mot, et en particulier les déterminatifs, doit être décrite en détail.

Il s'agit d'une description raisonnable pour la plupart des mots, mais pas pour tous. Certains sont écrits idéographiquement, mais ils ne sont pas très usuels dans les textes hiératiques. Plus ennuyeux est le problème des mots de fonction qui souvent ne suivent pas le schéma général. Dans ce cas, la chose raisonnable à faire est d'introduire des informations lexicales dans notre système et d'avoir des règles spécifiques pour ces mots.

ARCHITECTURE DU SYSTEME

Pour concevoir un système capable de traiter effectivement des textes, il fallait choisir un *corpus*. Les principes du système hiéroglyphique sont les mêmes pour tous les textes, mais les détails varient beaucoup. Par exemple, l'orthographe a considérablement changé de la XIIe à la XIXe dynastie. Même si l'on garde un corpus synchronique, l'orthographe des textes hiératiques ou des textes cursifs en général est beaucoup plus explicite que celle des textes monumentaux. Le système que nous sommes sur le point de décrire a été conçu pour les textes hiératiques du Moyen Empire.

Pour atteindre notre objectif, nous devons rendre le système un peu plus complexe. Nous aurons besoin non pas de deux, mais de cinq couches de règles. Chaque couche sera appliquée au résultat de la précédente. Toutes les couches sont implémentées en tant que transducteurs, mais certaines couches ne suivent pas le modèle ci-dessus.

Couches du système

Normalisation des signes

Cette première couche prend en entrée les codes MdC, et en sort une représentation simplifiée. Les *phonogrammes* du MdC sont remplacés par des codes Gardiner et certaines variantes de signes sont remplacées par leur signe de base (par exemple, F47  et F48  sont des variantes de F46  et seront donc remplacés par 'F46').

Valeurs de signe

Le niveau suivant de règles prend en entrée des codes normalisés et les remplace par toutes leurs interprétations possibles. Nous conservons les informations sur les signes bilitères, etc., afin de combiner les valeurs des signes dans la couche suivante.

Donnons quelques règles typiques, concernant le signe D36  :

D36 => P(a) / 100
D36 => ID(a) / 200
D36 => DET(action) / 500

D36 peut être compris comme un signe unilitère, qui est l'interprétation préférée (avec un coût de 100) ; mais il se comporte aussi comme un idéogramme, lors de l'écriture du mot *bras* ou du mot *condition, état*[7]. Le

[7] Dans ce dernier cas, il ne s'agit bien sûr pas d'un idéogramme *stricto sensu*, mais il se comporte comme tel.

coût de cette interprétation est légèrement plus élevé. L'idée est que, dans les règles ultérieures, la combinaison D36 et Z1 () sera interprétée comme un mot écrit idéographiquement à un coût relativement faible. La troisième règle traite d'une confusion entre D36 et D40  , comme déterminant des actions. Notez que cela aurait pu être traité dans le premier niveau à la place.

Dans la version actuelle du logiciel, nous distinguons différents types de valeurs :

- Valeurs phonétiques, codées par 'P'
- Valeurs idéographiques, codées par 'ID'
- Valeurs phonétiques-idéographiques, codées par 'IP'[8].

Certains signes sont aussi de véritables *mots-signes*, qui diffèrent de l'idéogramme usuel en ce qu'ils écrivent un mot tout seul, alors que l'idéogramme égyptien typique est généralement combiné avec Z1 pour ce faire (par exemple ). Dans ce cas, nous décidons de générer immédiatement l'interprétation complète du mot. Par exemple, O3,  , *pr.t-ḫrw* sera traité par la règle :

O3 => L(p), L(r), L(t), wordend, L(x), L(r), L(w), wordend / 100

où *wordend* est un marqueur pour les fins de mots. Ces données seront copiées telles quelles par les couches suivantes.

C'est aussi un bon endroit pour faire face à une sorte de hiatus qui se produit parfois dans le MdC. Par exemple, le groupe R22:R12  doit en fait être compris comme un tout, c'est-à-dire qu'il doit être considéré comme un seul signe[9]. Il en est de même pour le déterminatif T14-G41  , qui n'est pas un déterminatif *double*, mais représente en fait un oiseau frappé par un bâton, comme le montre la variante G84 : 

Enfin, comme certains signes sont des marqueurs assez forts de certains mots spécifiques, notamment des mots de fonction, mais peuvent être combinés avec des compléments phonétiques, nous avons décidé que, dans

[8] La pertinence de la catégorie de valeur phonétique-idéographique est quelque peu incertaine, comme me l'a souligné P. Vernus. Ils pourraient probablement être remplacés par des valeurs idéographiques.

[9] W. Schenkel « Gesichtspunkte für die Neugestaltung der Hieroglyphenliste », Göttinger Miszellen, vol. 14, 1974, p. 31–45.

la couche suivante, il pourrait être intéressant de conserver les codes des signes eux-mêmes, nous avons donc une *règle de copie* :

$X => $X / 0

Groupes simples

Cette couche prend son entrée de la couche précédente, c'est-à-dire les valeurs de signe, et combine ces signes. Elle ne produit pas encore de mots, car les mots peuvent être composés de groupes de signes multiples et assez complexes. Fondamentalement, ce niveau de règles fait deux choses :

a) il combine des compléments phonétiques avec d'autres signes, disant que 𓄿 mn u + C n est mn et non mnn (traitant des exceptions comme D4:D21 á qui est irr et non ir dans notre corpus).

b) il essaie de déterminer la fin des mots.

Concernant le cas a), on obtient en entrée des valeurs phonétiques comme P(m,n) ou P(n), des valeurs idéographiques, et des valeurs idéographiques-phonétiques. Nous devons les combiner pour réaliser les valeurs des groupes résultants. Cependant, comme nous nous intéressons à la formation des mots, nous devons garder l'information que toutes ces consonnes appartiennent au même groupe, car un groupe appartient à un seul mot. Ainsi, nos règles ont la forme :

P($X,$Y), P($Y) => G($X,$Y), groupend / 10

ce qui signifie qu'un signe phonétique de valeur XY, suivi d'un signe unilitère de valeur $Y doit être lu XY. Le fait que nous ayons regroupé ces signes est marqué par le symbole G(), ainsi que par le *groupend*, qui est utilisé dans les couches suivantes pour traiter la combinaison de groupes.

Concernant les terminaisons de mots, nous modélisons les combinaisons possibles de signes. L'idée est de représenter la façon dont les différents déterminatifs et marqueurs grammaticaux peuvent être combinés dans des mots. En particulier, un certain nombre de signes phonétiques, codant le pluriel, le féminin ou diverses valeurs verbales, peuvent être mélangés aux déterminatifs. Nous avons décidé de les réorganiser à ce stade, ce qui est pratique pour la couche suivante, où nous construirons le mot.

Examinons deux règles importantes à cet effet :

DET($x) => wordendstart, DET($x), wordend / 100
DET($x), P(t) => wordendstart, L(t), DET($x), wordend / 100

La première règle stipule qu'un déterminatif peut terminer un mot par lui-

même. La seconde prend un groupe composé d'un déterminatif et d'un 't', et déclare que cela peut être une terminaison de mot. Cependant, le 't' est positionné devant le déterminatif pour la couche suivante.

Je profiterai de cette règle pour indiquer pourquoi il est intéressant de distinguer les *phonogrammes* des *idéogrammes* dans notre système. Si l'on considère X2,  , qui pour notre époque est un idéogramme pour *t*, 'pain', il ne peut pas être utilisé comme marqueur phonétique pour les mots féminins. Et c'est exactement ce que fait notre système, car il n'y a pas de règle 'X2 => P(t)' dans le niveau *valeurs de signe*.

L'ensemble des règles de ce niveau est assez large, car nous devons traiter toutes les configurations possibles de signes. Sauf pour les plus évidentes, cela a été réalisé grâce à l'analyse du corpus.

La formation des mots

La tâche du prochain ensemble de règles est de combiner les différents groupes en mots. Il utilise les marqueurs créés par le niveau précédent.

Premièrement, les marqueurs *wordendstart* et *groupend* permettent d'exprimer qu'un mot doit en général contenir à la fois une partie phonétique et une partie déterminative (pour ceux qui n'en ont pas, nous avons des règles plus précises). Ainsi, nous avons une règle :

groupend, wordendstart => epsilon / 0

ce qui signifie qu'une fin de groupe suivie d'une partie *fin de mot* typique est normale, effaçant la séquence *groupend*, *wordendstart*, avec un coût de 0 (*epsilon* signifie *rien*).

groupend => wordend / 10000

ce qui signifie qu'une terminaison de groupe peut aussi être une terminaison de mot, mais à un coût très élevé en général. Les mots pour lesquels c'est normalement le cas sont peu nombreux et peuvent être traités à l'aide de règles spécifiques.

Ensuite, nous avons des règles pour combiner les différents groupes. Par exemple, un mot peut être écrit avec un groupe bilitère :

G($x,$y) => L($x), L($y) / 100

Ou il peut être écrit avec deux groupes unilitères :

G($x), groupend, G($y) => L($x), L($y) / 100

Dans cette dernière règle, la terminaison de groupe qui correspond au

premier groupe est *consommée* pour un faible coût (bien inférieur aux 10 000 nécessaires auparavant), ce qui signifie que le système préférera regrouper deux groupes unilitères dans un mot bilitère plutôt que de faire un mot avec le premier groupe seul (encore une fois, les mots fonctionnels comme le pronom suffixe ont des règles spécifiques qui raccourcissent celui-ci).

Les mots longs ne sont pas très courants en égyptien et ils ont tendance à suivre des modèles spécifiques. Par exemple, un mot peut être formé sur une racine bilitère en la redoublant, avec souvent un effet de renforcement. Par exemple la racine *ḳn*, *être fort* génère le verbe *ḳnḳn*, *battre*. Nous avons décrit cette formation avec la règle :

G($x,$y), e2, G($x,$y) => L($x), L($y), L($x), L($y) / 100

qui autorise les racines redoublées de la forme ABAB (les deux groupes doivent avoir les mêmes consonnes).

Déterminatifs phonétiques et nettoyage général

Le dernier ensemble de règles traite principalement des déterminatifs phonétiques. Ils sont actuellement considérés comme faisant partie de la fin du mot, mais ils apportent avec eux une valeur phonétique, qui devrait correspondre au début du mot.

Problèmes résolus

Fonctions et mots fréquents

Un certain nombre de mots, principalement des mots très courants, et surtout des mots de fonction, ont tendance à ne pas prendre du tout de déterminatif. Par conséquent, ils casseraient notre système.

Heureusement, bien que ces mots se retrouvent couramment dans les textes, ils ne représentent qu'une petite partie du lexique. Par conséquent, nous pouvons écrire des règles spécifiques pour eux. Dans le système actuel, ces règles sont conservées dans la couche *formation de groupe*, car dans cette couche nous avons à notre disposition les valeurs phonétiques des signes, et nous avons également copié leurs codes spécifiquement à cette fin.

La règle pour la préposition *ḥnꜥ* est :

P(H), P(n), P(a) => wordstart, L(H), L(n), L(a), wordend /1

Ici, *wordstart* et *wordend* empêcheront toute tentative de grouper ce mot avec un autre plus tard.

Cependant, il y a quelques problèmes, en particulier avec les pronoms suffixes, et en particulier avec 's', qui peut être soit un pronom suffixe féminin, soit être utilisé comme préfixe causal. Un point qui doit certainement être amélioré dans notre système est qu'il sort parfois une séquence de suffixe, ce qui est évidemment impossible. Il serait possible de créer un transducteur pour empêcher ce comportement, mais cela n'a pas encore été fait.

Déterminatifs multiples

Dans l'orthographe moyen-égyptienne, les mots ont tendance à n'avoir qu'un seul déterminatif, mais ils en prennent parfois plus. Comme nous avons décidé d'extraire nos règles du corpus, nous avions besoin de lister ces cas. En fait, ils ne sont pas arbitraires. L'un de ces déterminatifs tend à être très général, comme Y1  (abstractions), A1  (homme), ou N35A  (liquide), l'autre étant beaucoup plus spécifique et, dans certains cas, on pourrait être tenté de le considérer comme un idéogramme. Par exemple, dans le mot  , *ktt*, *petit*, le premier déterminatif () correspond aux petites choses, tandis que le second correspond aux femmes.

Nous avons décidé d'ajouter une marque, 'GENDET', à ces signes. Une règle de la forme :

DET($X), wordend, GENDET, DET($Y) => DET($X), DET($Y), wordend / 1

permet alors la combinaison d'un déterminatif générique avec un autre, tout en empêchant la combinaison de deux déterminatifs banals.

Robustesse

Un logiciel est dit robuste s'il peut faire face à des entrées *mauvaises* ou inattendues. Le principe même de notre système permet de le faire assez facilement.

L'idée est que chaque couche doit accepter toute entrée possible, mais toujours produire une sortie. Cela se fait par des règles *fourre-tout*, qui ont un coût très élevé.

CONCLUSION

Évaluation

Nous avons appliqué nos règles sur le *Conte du naufragé*[10] (3 500 signes, 1 419 mots), et nous avons obtenu environ 9% de mots erronés. Un mot était considéré comme erroné si l'interprétation donnée par le système n'était pas possible ; dans certains cas, il pourrait être considéré comme quelque peu optimiste car le contexte grammatical pourrait par exemple invalider l'analyse du logiciel.

Le logiciel a ensuite été essayé sur un autre texte, le *Papyrus Westcar* (9 480 signes, 4 000 mots), pour lequel des règles n'avaient pas été construites. Le taux d'erreur est passé à 18%, ce qui est plutôt élevé. Cependant, la plupart des erreurs ont été causées par des groupes de signes imprévus, ce qui est plutôt facile à corriger dans notre formalisme.

Une bonne caractéristique du système est que l'on peut ajouter des règles complexes assez librement, sans interférer avec les parties déjà correctes du logiciel, car une règle complexe correspond à une entrée complexe et particulière, et n'interférera donc avec des règles plus générales que dans des cas spécifiques, ce qui devrait normalement améliorer le résultat. Bien sûr, les règles qui acceptent des entrées très simples sont définitivement plus difficiles.

Or, si l'on prend un texte de l'égyptien tardif, très éloigné du corpus de formation, le résultat est plutôt mauvais mais assez intéressant. Notre système, lorsqu'il essaie de translittérer l'*Enseignement d'Amenemopet*[11], fait le même genre d'erreurs qu'un étudiant formé en moyen égyptien ferait. En particulier, il est déconcerté par l'orthographe syllabique et les déterminatifs multiples. D'une certaine manière, cela indique que nous avons modélisé la performance d'un élève formé à la langue classique, mais pas celle de ses successeurs plus récents.

D'un côté plus pratique, le logiciel est assez rapide sur un ordinateur moderne. La vitesse est actuellement de l'ordre de 30s pour 3 000 signes, ce qui semble raisonnable (et bien plus rapide qu'un humain n'y parviendrait !). Quelques optimisations assez simples permettraient d'améliorer considérablement les résultats.

Améliorations possibles

Un certain nombre de points techniques peuvent certainement être améliorés dans le système ; en particulier, la couche de valeur de signe

[10] The Tale of the Shipwrecked Sailor.
[11] The Instruction of Amenemope.

pourrait bénéficier d'une augmentation considérable de la vitesse. Le système de règles pourrait probablement être amélioré, à la fois en ajoutant quelques couches (en particulier des couches pour filtrer plusieurs déterminatifs), et en nettoyant celles existantes. Le système de coût lui-même est bien sûr très *ad-hoc*, et il serait intéressant de voir si les statistiques pourraient être appliquées à ce problème[12], mais cela nécessiterait une révision majeure du système, avec une structure beaucoup plus simple. L'inconvénient des statistiques dans ce cas est que nous ne serions plus en mesure de modéliser l'expertise humaine.

Le système dans son ensemble est plutôt indépendant du corpus de formation proprement dit, et on pourrait imaginer utiliser différents ensembles de règles pour différents types de textes : textes hiéroglyphiques, textes égyptiens tardifs, etc.

Bien sûr, l'ajout d'un lexique augmenterait la précision du système, et c'est en effet assez facile, comme en témoignent les travaux sur les mots de fonction.

Utilisations possibles

Notre système peut bien sûr être utilisé pour produire une translittération *raisonnablement bonne* d'un texte, ce qui peut accélérer le travail d'un chercheur (bien que nous ne soyons pas sûrs que corriger une translittération existante soit beaucoup plus rapide que d'en écrire une à partir de rien).

Le résultat lui-même, un transducteur, peut être utilisé comme entrée pour un traitement ultérieur, par exemple comme entrée d'un analyseur syntaxique.

Une propriété intéressante du système est qu'il analyse la structure du mot tout en le translittérant. Cela pourrait être utile, par exemple dans les recherches de texte. Les niveaux intermédiaires, avec le groupe de signes, peuvent être considérés comme des *formes normalisées* des mots, et pourraient être utilisés pour la recherche et l'indexation.

Et, *last but not least*, la création de règles en elle-même est plutôt stimulante et consiste en un exercice assez intéressant dans l'étude de l'orthographe égyptienne antique.

[12] F. Pereira, M. Riley. Reconnaissance de la parole par composition d'automates finis pondérés . Dans Roche & Schabes, éd., p. 431–453.